NAVAL POSTGRADUATE SCHOOL

Monterey, California

THESIS

AN ANALYSIS OF MISSION CRITICAL COMPUTER SOFTWARE IN NAVAL AVIATION

by

Robert L. Buckley

March 1991

Thesis Advisor: Martin J. McCaffrey

Approved for public release; distribution is unlimited

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) Code 36	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS Program Element No / Project No / Task No / Work Unit Accession Number

11. TITLE (*Include Security Classification*)
AN ANALYSIS OF MISSION CRITICAL COMPUTER SOFTWARE IN NAVAL AVIATION

12. PERSONAL AUTHOR(S) Buckley, Robert L.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*year, month, day*)	15. PAGE COUNT
Master's Thesis	From To	March 1991	78

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (*continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUBGROUP	Software Engineering, Software Maintenance, Software Development, MCCR, ECR, Aviation Software, Naval Aviation Software, Military Software Regulations and Standards.

19. ABSTRACT (*continue on reverse if necessary and identify by block number*)

For over 25 years, the United States Navy has been designing, developing and maintaining software for embedded computer systems. Throughout this generation of Naval aviation software development, no collective analysis of the successes and failures in software development had been accomplished. To accomplish this task, this thesis evaluated aircraft software data from the Department of the Navy against two metrics: 1) did the original software development schedule have to be changed, and 2) did the software released to the fleet contain any major defects? This research has revealed that only about half of the original software development schedules were sustained without a milestone change being made. Also, software that was released to the fleet had no major deficiencies three out of four times. To further specify this information, it has been refined into categories of software language, size of program and type of software program. The results of this study will be beneficial to aviation program managers, software developers and software maintenance technicians

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Martin J. McCaffrey	22b. TELEPHONE (*Include Area code*) (408) 646-2488	22c. OFFICE SYMBOL AS/Mf

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

Approved for public release; distribution is unlimited.

An Analysis of Mission Critical Computer Software in Naval Aviation

by

Robert L. Buckley
Lieutenant Commander, United States Navy
B.S., Texas A&M University, 1979

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March 1991

ABSTRACT

For over 25 years, the United States Navy has been designing, developing and maintaining software for embedded computer systems. Throughout this generation of Naval aviation software development, no collective analysis of the successes and failures in software development had been accomplished. To accomplish this task, this thesis evaluated aircraft software data from the Department of the Navy against two metrics: 1) did the original software development schedule have to be changed, and 2) did the software released to the fleet contain any major defects? This research has revealed that only about half of the original software development schedules were sustained without a milestone change being made. Also, software that was released to the fleet had no major deficiencies three out of four times. To further specify this information, it has been refined into categories of software language, size of program and type of software program. The results of this study will be beneficial to aviation program managers, software developers and software maintenance technicians.

TABLE OF CONTENTS

I. INTRODUCTION

A. GENERAL

The use of computers and application software has become a part of everyday life in today's society. The same can be said for military organizations in all areas of specialization (i.e. administrative, training, maintenance, operations, etc). However, the area where the Department of Defense (DOD) spends the most money for computer software is in the area of embedded computers [Ref. 1]. The one section of embedded computers where the user requirements are the most stringent is in weapons systems, especially aviation systems where decisions have to be made instantaneously with little or no room for error. The United States Navy has been designing, developing and implementing software for embedded computer avionics systems for over 25 years, but as the requirements of computer systems have become more demanding and aircraft have become more sophisticated the problems with software have increased (at least observably so). The difficulties of software projects meeting the original schedule have been noted throughout the computer industry and DOD is not immune from this problem. "... Air Force General Bernard Randolph, chief of Air Force Systems Command, characterized software as the Achilles' heel of weapons development. "On software

schedules, we've got a perfect record: We haven't met one yet,...." [Ref. 2] Additionally, this problem is magnified for DOD aviation programs because of their tremendous size, complicated specifications, large budgets and high public visibility. But, do all Navy aviation software programs have a problem with meeting their schedule or it is just the publicized ones who get the notoriety? And, if software programs have problems being on-time is it generally for the same cause or are there different reasons for the delays? This thesis will answer these questions and analyze the software factors which cause problems before and after a program is released to the Fleet.

B. SCOPE

In an attempt to discover why some software projects are successful and others are not, an analysis was conducted of the life cycle management of Naval Aviation mission critical software. This analysis of the life cycle management of mission critical software will be made comparing current and historical data on the software which operates the mission computers of several aviation platforms in the fleet.

C. METHODOLOGY

This thesis was developed using a four step approach. First, a general idea of research interest was determined and later was more narrowly focused to fit research capabilities.

Second, a thorough literature review and data search was conducted to discover past efforts in this area, in an effort to develop a working database. Next, questionnaires, field trips and phone conversations were conducted to gain as much specific information on each project as possible. Finally, all data was collected and analyzed allowing conclusions and recommendations to be made.

D. FOCUS

This research was designed to collect as much information on embedded computer systems and their software in Navy aircraft as possible. Due to the many different types of computer systems in aircraft and the difficulty in accomplishing a valid comparison between systems only one type of computer system was chosen to collect data about. The system which almost all aircraft have in common is the main or mission computer. The software that runs these mission computers is an Operational Flight Program (OFP) or equivalent. The following Navy and Marine Corps aircraft use OFPs or the equivalent and are included in this research:

A-6E	AH-1W
AV-8B	EA-6B
E-2C	F-14A
HH-60H/J	P-3C
S-3A/B	SH-2

SH-3 SH-60B

SH-60F

E. ORGANIZATION

Chapter II provides background information for this thesis. The applicable regulations and guidelines that Navy program managers must follow in developing, implementing and maintaining an embedded computer system are summarized. Additionally, a more thorough explanation of the principal document used in embedded software development is given.

The process of data collection for this research is explained in Chapter III. The unique terms for software development are defined along with the metrics used to compare software projects. An explanation of the inquiry used to collect the data is given in an effort to show what information was required.

Chapter IV contains an explanation of how the data was analyzed. The process of data comparison and analysis is shown along with the numerical results obtained from this process.

The conclusions and recommendations are contained in Chapter V. A more in-depth explanation of the results obtained in Chapter IV is given, plus other noteworthy facts collected during this research. Additionally, a section outline with possible follow-on topics from this area of research is included.

II. REGULATIONS FOR SOFTWARE DEVELOPMENT

A. OVERVIEW

All Navy program managers (PMs) who are in charge of major defense systems that contain embedded computer resources must observe set standards and guidelines for software development. Each regulation is written to integrate all phases of military software life cycle management and covers either overall policy or specific details for software development. To more fully understand, the purpose of each regulation and what information a PM or software developer can obtain from them, a short synopsis of these major standards and guidelines is provided.

1. DOD Directive 5000.29

Published in 1976, this directive establishes policy for the DOD in the management and control of the development, acquisition, deployment and support of computer resources in major defense systems. This directive requires embedded computer resources to undergo validation and risk analysis, configuration management, and life cycle planning. To oversee and coordinate the policies of this directive the Management Steering Committee for Embedded Computer Resources (MSC-ECR) was created. Besides improving the management of embedded computer resources in major defense systems, this committee

works to ensure that new computer research, development, technology and policy are a part of normal defense system acquisition process. DOD Directive 5000.29 is the basis of all other Department of the Navy (DON) instructions on managing embedded computer resources.

2. DOD-STD-2167A

This DOD standard is the keystone regulation for the entire software development process with all other regulations providing either implementation policy or support for specific phases of life cycle management. This standard sets the requirements to be used during acquisition, development and support of mission critical software systems. These software life cycle requirements are not only mandatory for DOD agencies but also for the contractor. The major phases for the software development process that 2167A recommends will be explained in more detail below.

3. DOD-STD-2168

This DOD standard works in conjunction with DOD-STD-2167A to establish the requirements for a software quality program. To fulfill these requirements, a process must be implemented to effectively resolve software problems by evaluating software quality, documentation and related activities in a timely manner. The requirements of this standard are applicable to DOD agencies and contractors during the entire software life cycle from acquisition to support.

4. **DOD-STD-1679A**

This DOD standard is the predecessor to DOD-STD-2167 and lists the original DOD requirements for mission critical software development. This software development procedure was written to allow for changing operational requirements, reduction of life cycle costs and the highest degree of software reliability and maintainability. Since this was the software development standard before September 1986, most Naval aviation software programs in operation today are covered under this standard.

5. **DOD-HDBK-287**

This DOD handbook was published to assist government agencies in tailoring DOD-STD-2167A for either a software development contract or a software support contract. The handbook provides key concepts of DOD-STD-2167A and the factors that should be considered when tailoring a software contract to this DOD standard.

6. **MIL-STD-480B**

This military standard sets forth the requirements and procedures for configuration control in the acquisition of software items. An important part of configuration control is the correct process of making changes to an already approved configuration item. The documents used for requesting these changes are known as Engineering Change Proposals (ECPs), deviations or waivers. The procedures, formats and rules for

submitting these documents for changes are provided in MIL-STD-480B to standardize this process.

7. **MIL-STD-1521B**

This military standard provides the requirements to be followed when conducting technical reviews and audits of computer systems and software. This standard lists general and specific requirements that both the contracting agency and the contractor must accomplish during each phase of review or audit. Like DOD-STD-2167A, this standard shall be tailored to use only the applicable requirements for the computer resource being acquired.

8. **SECNAVINST 5200.32**

This Secretary of the Navy Instruction (SECNAVINST) sets DON policy for managing embedded computer resources including software. The overall policy of this instruction is to ensure that all levels of DON project and acquisition management give proper emphasis to life cycle and software management, risk and cost analysis, and stabilization of computer resource requirements. This instruction supplements the policies and procedures of DOD Directive 5000.29.

9. **OPNAVINST 5200.28**

This Chief of Naval Operations Instruction (OPNAVINST) establishes the CNO policy for life cycle management of Mission-Critical Computer Resources (MCCR) under the Research, Development and Acquisition (RDA) process. This policy covers

all MCCR including software that are an integral part of or are used in support of weapons systems. The purpose of this instruction is to ensure that all MCCR that support weapons systems are integrated into the same life cycle management process as the weapons system. This life cycle management process begins from the very start of the acquisition process and continues through the post-deployment software support (PDSS) phase.

10. **NAVELEXINST 5200.23**

This instruction, which was originally promulgated by the Naval Electronic Systems Command (NAVELEX) and now is administered by the Commander, Space and Naval Warfare Systems Command (COMSPAWARSYSCOM), is the current U.S. Navy guide on computer software life cycle management. Information on software engineering and life cycle management of the software acquisition process is provided for use by program managers. This instruction also provides some of the factors that are common software problems, how current DOD policies were established to respond to these problems and why each phase of computer software life cycle management is important.

11. **TADSTANDS A through E**

These Tactical Digital Standards (TADSTANDS) A through E, which are administered by COMSPAWARSYSCOM for the Navy, establish the standards to be used during system development and life cycle support. Each TADSTAND sets the policy or

requirements for the standardization of one of five areas: definitions for embedded computer resources (ECR), computer interface devices, programming and design languages, reserve computer capacity and requirements for mission-critical systems software acquisition, development and support.

B. DOD-STD-1679A AND DOD-STD-2167A DIFFERENCES

As the two main standards that are used to guide MCCR development, it is important to understand what changes, if any, were made between the first standard DOD-STD-1679 and it's successor DOD-STD-2167. One of the basic differences between the two standards is that DOD-STD-2167A is more current with computer technology and refers to the components of software as Computer Software Configuration Items (CSCI) while DOD-STD-1679A uses older terminology and refers to software components as programs, subprograms, modules and units. The only software programs that are subject to DOD-STD-2167 are those which have either issued a request for proposal for full scale engineering development (FSED) or entered FSED after September 1986. Although, the spirit and intent of both DOD-STD-1679A and -2167 are very similar, both standards approach software development differently. This can be seen when comparing the detailed requirements of DOD-STD-2167A (explained in the next section) and the detailed requirements of DOD-STD-1679A listed below:

1. Software Performance Requirements
2. Software Design
3. Programming Standards
4. Programming Conventions
5. Software Implementation
6. Software Generation
7. Software Operation
8. Software Testing
9. Software Quality Assurance
10. Software Acceptance
11. Software Configuration Management
12. Software Management Planning

C. DETAILED REQUIREMENTS OF DOD-STD-2167A

Since DOD-STD-2167A (from now on referred to as 2167A) is the current MCCR software development standard for DOD, it will be explained in more detail than DOD-STD-1679A (from now on referred as to 1679A). Again, the spirit and intent of these two instructions is virtually the same -- development of the best quality software. 2167A establishes specific software development management requirements which must be followed by DOD contracting agencies and contractors. However, the standards can be tailored by the contracting agency if a requirement is non-applicable. The tailored set of requirements for each software program will be specified in

the contract agreed upon by the contractor and the contracting agency. All detailed requirements prescribed by 2167A contain elements of the general requirements: software development management, software engineering, formal qualification testing, software product evaluation and software configuration management. The standard set of detailed requirements are as follows:

1. **System requirements analysis/design**

This section of 2167A requires the software contractor to conduct a thorough analysis of system specifications for consistent and complete software requirements, and to optimize computer resource allocations (i.e. hardware, software and personnel). Also, the contractor shall support all system reviews as specified in the contract, plus evaluate and collate by specified criteria the proper preliminary documentation.

2. **Software requirements analysis**

The software contractor is required to conduct reviews of the software specifications by the standards set in MIL-STD-1521 (from now on referred to as 1521) and to establish the baseline for the Computer Software Configuration Items (CSCI). All engineering, interface and qualification requirements for each CSCI shall be documented by the contractor, and evaluation of software and interface requirements must also be performed by criteria set in 2167A.

3. Preliminary design

The Preliminary Design Review (PDR) of the software is to be conducted by the contractor according to the procedures set forth in 1521. The PDR ensures that the following items have been developed: a Software Design Document (SDD) which contains separate preliminary designs for each CSCI and requirement allocations; a Software Test Plan (STP) for formal qualification tests for each CSCI; and a preliminary Interface Design Document (IDD) which contains the preliminary design of interfaces external to each CSCI.

4. Detailed design

The Critical Design Review (CDR) of the software is to be conducted by the contractor according to the procedures set forth in 1521. The CDR is more specific than the PDR. The CDR verifies that the detailed design for each CSCI has been accomplished and documented in the SDD and IDD. The CDR also verifies that specific test cases have been described for each formal qualification test and documented in the Software Test Description (STD).

5. Coding and CSU testing

This section of 2167A requires the contractor to code and test each Computer Software Unit (CSU) to ensure specified requirements are meet. If changes are necessary to the CSU code, then revisions to the design, documentation and code

will be made by the contractor along with any necessary retesting.

6. CSC integration and testing

After CSU coding and testing is complete, then the CSUs are assembled together into the correct Computer Software Component (CSC). The contractor must integrate and test each CSC to ensure specified requirements are meet. If changes are necessary, then revisions to the design, documentation and code will be made by the contractor along with any necessary retesting of CSUs or CSCs. A Test Readiness Review (TRR) will be conducted by procedures set forth in 1521 to ensure the CSC integration and testing is complete.

7. CSCI testing

This section of 2167A software development is where the Formal Qualification Testing (FQT) is conducted for each CSCI. If changes are necessary due to FQT results, then revisions to the SDD, IDD and code will be made by the contractor along with any necessary retesting of applicable CSU, CSC or CSCI. The STD sets the procedures to be used for the FQT with the results being recorded in a Software Test Report (STR). The contractor may also support the Functional Configuration Audit (FCA) and Physical Configuration Audit (PCA) if conducted during this section.

8. **System integration and testing**

The contractor will support all areas of system integration and testing and make any revisions to documentation and coding including retesting as necessary. Additionally, if FCA and PCA are conducted during this section, then the contractor will support it.

This total process is graphically displayed in Figure 1 which is reproduced from DOD-STD-2167A:

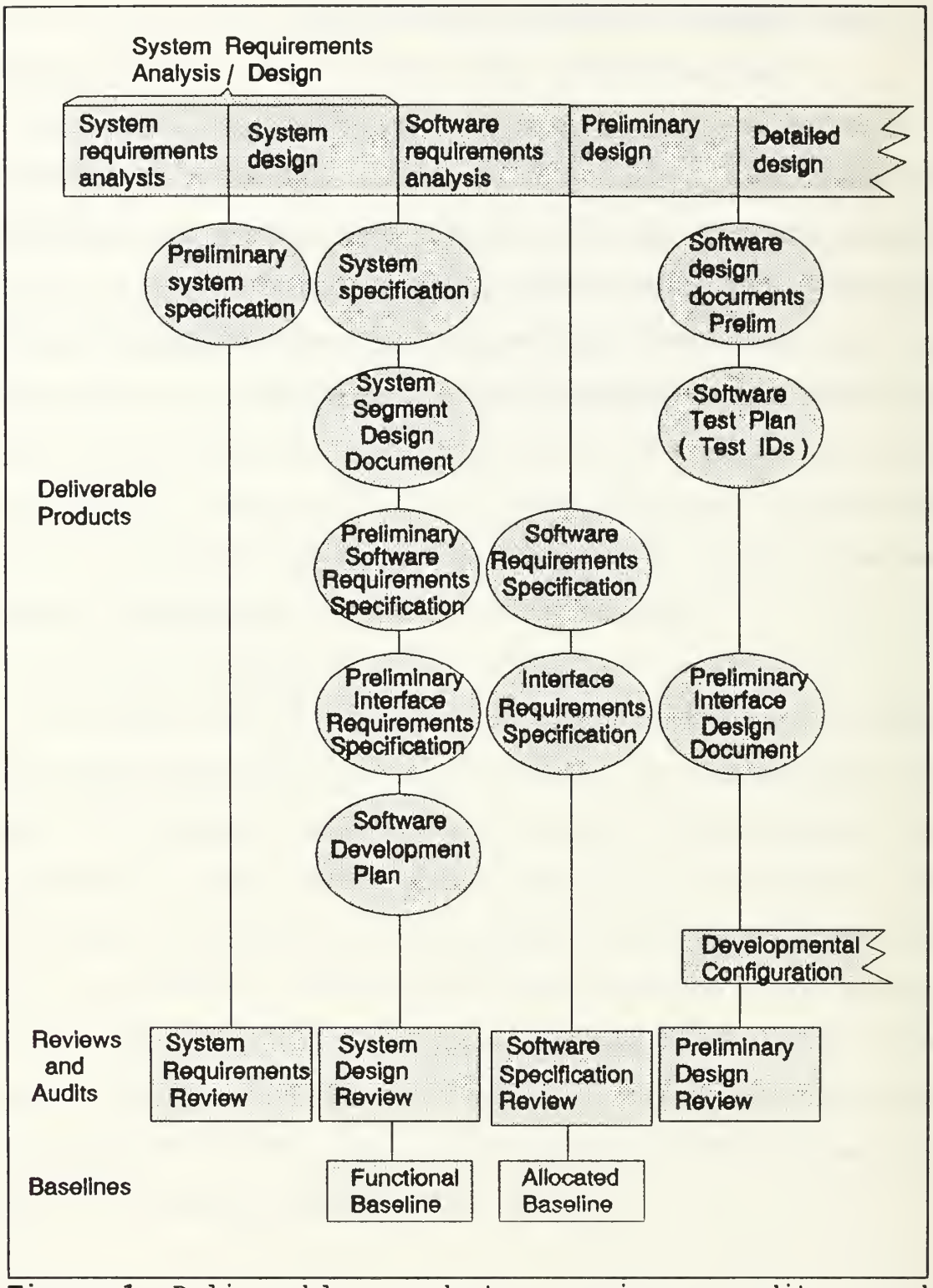

Figure 1 Deliverable products, reviews, audits, and baselines.

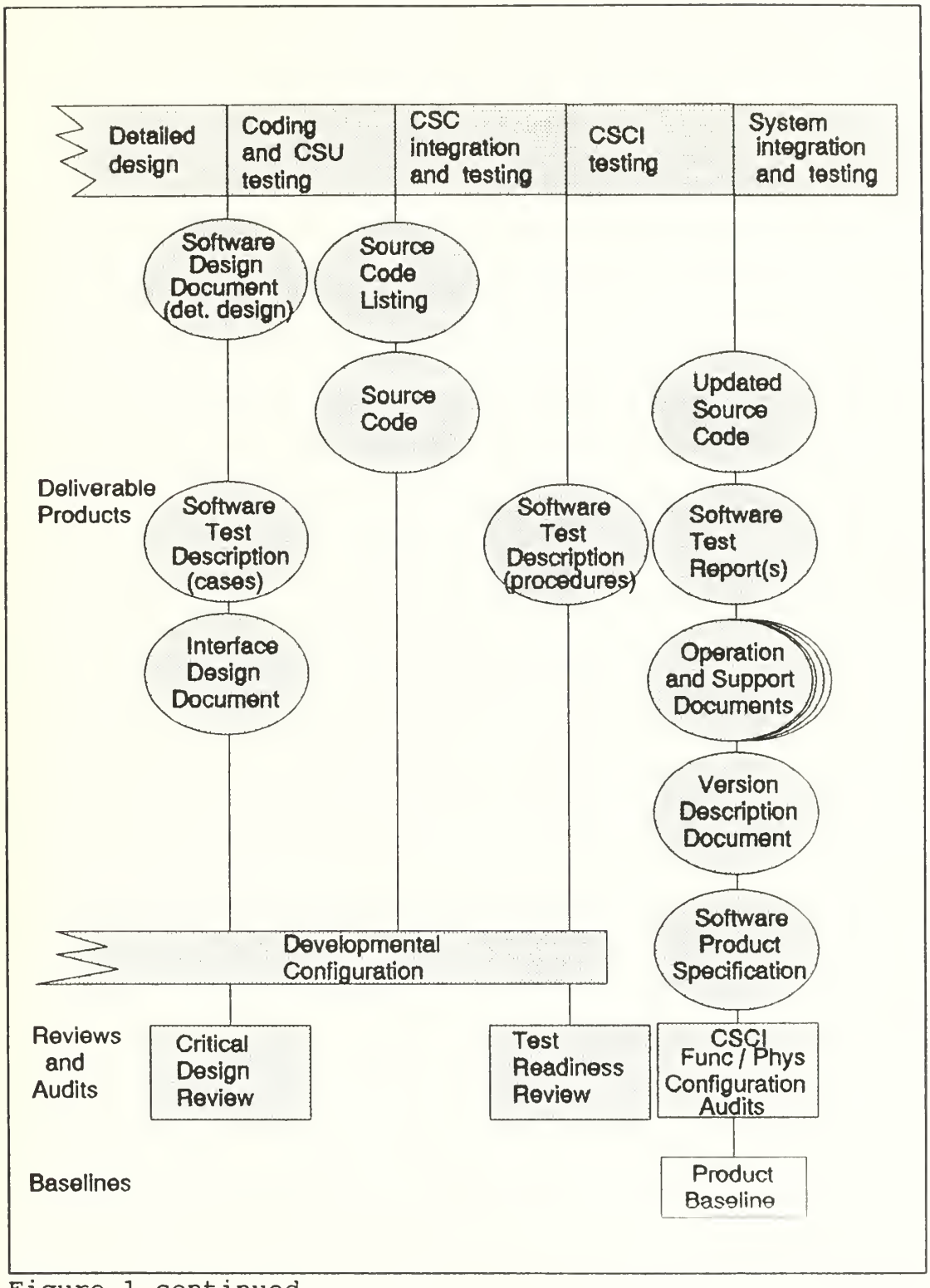

Figure 1 continued

III. DATA COLLECTION

A. DEFINITION OF TERMS

The following words and terms are defined from DOD-STD-2167A, MIL-STD-1521B, MIL-STD-480B, or TADSTANDS A and D:[Ref. 3]

1. Computer Resources

"The totality of computer equipment, computer programs, computer data, associated documentation, personnel, and supplies."

2. Computer Software Component (CSC)

A distinct part of a computer software configuration item (CSCI). CSCs may be further decomposed into other CSCs and Computer Software Units (CSUs).

3. Computer Software Configuration Item (CSCI)

"A configuration item for computer software."

4. Computer Software Unit (CSU)

"An element specified in the design of a Computer Software Component (CSC) that is separately testable."

5. Configuration Item (CI)

"An aggregation of hardware, firmware, software, or any of its discrete portions, which satisfies an end use function and is designated for configuration management."

6. Critical Design Review (CDR)

This review shall be conducted for each configuration item when detail design is essentially complete. For CSCIs,

this review will focus on the determination of the acceptability of the detailed design, performance, and test characteristics of the design solution, and on the adequacy of the operation and support documents.

7. Embedded Computer (EC)

A digital computer or processor that is an integral component, from the design, procurement, and operations point of view, of any tactical digital system. This definition includes microcomputer, microprocessor, etc.

8. Embedded Computer Resources (ECR)

The totality of operational and support software/firmware; embedded computers; data storage and display devices; interface standards; programming languages; support facilities ashore; training facilities; training support personnel; and personnel whose primary specialized educational experience and/or training is directed toward operation or maintenance of embedded computers. Specifically included are programmable calculators (PROCALS) that are electrically interfaced to tactical digital systems.

9. Formal Qualification Testing (FQT)

"A process that allows the contracting agency to determine whether a configuration item complies with the allocated requirements for that item."

10. Functional Configuration Audit (FCA)

A formal audit to validate that the development of a configuration item has been completed satisfactorily and that the configuration item has achieved the performance and functional characteristics specified in the functional or allocated configuration identification. In addition, the completed operation and support documents shall be reviewed.

11. Mission-Critical Computer Resources (MCCR)

Computer resources acquired for use as integral parts of weapons; command and control; communications; intelligence; and other tactical or strategic systems aboard ships, aircraft, and shore facilities and their support systems. The terms also includes all computer resources associated with specific program developmental

test and evaluation, operational test and evaluation, and post-deployment software support including weapon system trainer devices, automatic test equipment, land-based test sites, and system integration and test environments.

12. Physical Configuration Audit (PCA)

"A technical examination of a designated configuration item to verify that the configuration item 'As Built' conforms to the technical documentation which defines the configuration item."

13. Preliminary Design Review (PDR)

"For CSCIs, this review will focus on: (1) the evaluation of the progress, consistency, and technical adequacy of the selected top-level design and test approach, (2) compatibility between software requirements and preliminary design, and (3) on the preliminary version of the operation and support documents."

14. Problem Reports

Also referred to as Software Trouble Reports (STRs) or Program Trouble Reports (PTRs) -

Problems detected in the software or its documentation shall be classified by priority as follows:

a. Priority One

(Also referred to as an Emergency PTR) - A software problem that does one of the following:

(1)Prevents the accomplishment of an operational or mission essential capability specified by baseline requirements

(2)Prevents the operator's accomplishment of an operational or mission essential capability.

(3)Jeopardizes personnel safety.

b. Priority Two

A software problem that does one of the following:

(1)Adversely affects the accomplishment of an operational or mission essential capability specified by

baselined requirements so as to degrade performance and for which no alternative work-around solution is known.

(2)Adversely affects the operator's accomplishment of an operational or mission essential capability specified by baselined requirements so as to degrade performance and for which no alternative work-around solution is known.

15. Test Readiness Review (TRR)

A review conducted for each CSCI to determine whether the software test procedures are complete and to assure that the contractor is prepared for formal CSCI testing. Software test procedures are evaluated for compliance with software test plans and descriptions, and for adequacy in accomplishing test requirements.

B. METRICS USED

The success of any software project can only be obtained from the standards by which it is judged. For this analysis, the metrics used to define a successful software program were:

1. Did the program meet the initial planned delivery date with the specified software requirements?

2. Was the program operationally successful (i.e. no priority one or two Problem Reports were issued after the software was released to the fleet)?

Other factors which must be considered in this comparison analysis are program size, computer language the program is written in, and type of program (i.e. new, an upgrade to an existing program or a maintenance fix for problems reported in a previous program).

C. DATA SEARCH

The collection of data in any research is difficult, especially if there is not a place acting as a central data repository. Another factor is the political sensitivity of the data to the responsible organization. These two factors are true for software data collection in both the public or private sectors and understandably so. In searching for data sources for this research, the few possible software databases which might contain viable information could not be used because of data confidentiality. Since no current databases could be used or found, an office in the Naval Aviation Command (NAVAIR) with connections to all aviation computer systems was discovered which could be a central point for data collection. However, because of the immense amount of background information that was needed on aircraft software, the best source of information was decided to be from the Software Support Activities (SSAs) in the field. The SSA is an organization whose purpose is to provide software maintenance and support for one specific aircraft type after the software contractor has completed contractual obligations and delivered the software. The SSA also monitors all phases of software development by the contractor and has the most complete records on aircraft software development of any Navy organization. Using all possible sources of information was important; therefore, the sources of data for this research

have come from offices in NAVAIR, the SSAs of the aircraft types used and technical support contractors.

D. INQUIRY BACKGROUND

1. Purpose

A major difficulty in collecting data from the SSAs is due to the fact that they are not in one central location, but are dispersed across the United States. In order to collect the research data necessary, an inquiry or small questionnaire was developed to collect the information required for this analysis. The inquiries were mailed or faxed to the 11 SSAs which participated in this study. Each inquiry was followed up with telephone conversations to alleviate any of the questions that either side may have had about the information being requested or the data being supplied. The inquiry data collection procedures were continually updated to ensure conciseness, clarity and completeness. A copy of the final revision of the inquiry used for data collection is provided in Appendix A.

2. Information Collected

As shown in Appendix A, the inquiry collected information on specific items about the software, as follows:

1. Type of aircraft.
2. Type of computer system.
3. Total number of Lines of Code (LOC) in software program today.

4. Computer language that the software program was written in.

5. A copy or list of the Software Life Cycle Schedule (sometimes referred to as "Milestone Charts")for each version of the software program, in order to gain a pictorial perspective of the history of each software program.

6. For each software version, an explanation of why schedule changes, if any, (noted in 5 above) were made.

7. Number of LOC which were new or changed from one version of the software program to the next.

8. Type of software program each version was (e.g. Initial, Upgrade, Maintenance or Other).

9. For each software version, reasons for Priority one or two Problem Reports, if any.

10. Space for additional comments on aviation software development, specific or general.

3. Definition of Inquiry Terms

Some of the terms in the software field are vague and not well defined. To help alleviate this situation, the following words and terms used in the inquiry are more fully defined.

a. Lines of Code (LOC) - executable statements and data definitions are counted, but not comment statements and headers.

b. Software Type - classification of the purpose of the software version released to the fleet.

(1) Initial - the original release of a software program for a new aircraft type or configuration, or for a major change in computer hardware configuration.

(2) Maintenance - the release of a software program to correct the problems (priority one or two PTRs) of a previously released software version.

(3) Upgrade - the release of a software program to enhance the capabilities of or provide new features to a current working software version. Upgrades may also contain some corrections to minor software problems from previous releases.

(4) Other - any software release which does not fall under the three classes above.

c. Software Version - the nomenclature used to differentiate between a previous software program and any changes or updates made to that previous software program (e.g. program A will come before program A.1 or program B).

IV. DATA ANALYSIS

Although the information collected in this research may have been specific as to personnel and organizations, the results of this study will only be of a generic nature. The categorizing of data into specific software factors was done to provide useful information on software development without drawing attention to certain software programs. This high level of confidentiality of source data was established in order to obtain the most accurate and candid information.

A. DATA PARAMETERS

Chapter III Section B discussed the metrics used to determine a successful software program, while Section D discussed the information that was collected in the inquiry. From the data collected, it has been reported that when a milestone changed for a software version, it was never accelerated but instead was always delayed. It has also been reported that once a milestone was delayed that the entire software development schedule was also delayed including the software delivery date. Therefore, if the inquiry data reported that a milestone change had occurred then the answer to metric standard one, about the initial planned delivery date being on time, was NO. Similarly, the metric standard about program operational success was YES if the program had

no priority one or two Program Trouble Reports (PTRs) and NO if there were priority one or two PTRs. These two metric standards, milestone changes and priority one or two PTRs, were compared against the total number of software versions in the study. To provide more precise information, the total number of software versions were changed into more specific technological categories of 1) software language, 2) program size, and 3) software type. Each of these categories was then subsequently divided into more explicit subcategories in order to further refine the results as follows:

1) The software language category was divided into assembler and CMS-2, the two languages used for all programs in this study.

2) The program size category was separated into three subcategories of programs in the size ranges: 0 - 90,000, 90,000 - 200,000, and 200,000 and above bytes.

3) The software type category, as previously defined in Chapter III Section D, was subdivided into initial, upgrade, maintenance and other.

Also, the reasons why these milestones changed will be given along with percentage of occurrence.

B. METHODS OF DATA ANALYSIS

The analysis techniques that were determined to be appropriate for this study were the relative frequency of

class data or percentages and the Chi-square independence test.

1. Frequency of Class Data Analysis Method

To accomplish the first analysis technique of frequency of class data, the data on each success metric, milestone change and priority one or two PTRs, was divided into YES and NO responses for each metric. These four numbers were then divided by the total number of software versions in the study in order to get the overall percentage of software versions which were not delayed, delayed, operationally successful or not operationally successful. Each software version was then viewed from the more technical categories of: software language, program size or software type. These categories were further refined into their respective subcategories so as to make the data more specific. First, a percentage of each subcategory was calculated by adding up the total number of software versions per subcategory and dividing this number by the total number of software versions. Next, each subcategory was divided into YES/NO responses for each success metric, milestone change and priority one or two PTRs, and a percentage was calculated. This was accomplished by totaling up all the subcategory software versions that did and those that did not have milestone changes (those that had priority one or two PTRs and those that did not) and dividing this number by the total number of software versions in that

subcategory. Finally, a percentage was calculated for each different reason for why milestones changed. To accomplish this, the total number for each reason was divided by the total number of overall reasons.

2. Chi-square Independence Test Analysis Method

To accomplish the second analysis technique, each category (software language, program size and software type) was tested for independence with each of the two metrics (milestone change and priority one or two PTRs) using the chi-square independence test. Additionally, the two metrics were tested for dependency with each other. The chi-square independence method tests two events for statistical independence which is defined as: "... if the occurrence (or nonoccurrence) of one of the events does not affect the probability of the occurrence of the other event." [Ref. 4] The term independence will be used in place of statistical independence. The chi-square independence test requires that null and alternative hypothesis be stated.

The null hypothesis is that each category was independent of each of the two metrics, and the alternative hypothesis was that each category was dependent of each of the two metrics. The chi-square procedure uses the observed values for each sub-category as shown in Tables 1-3, and a calculated expected value to derive the chi-square value for testing. The expected value is calculated by multiplying the

row total of software versions by the column total of software versions and dividing by the total number of software versions for each sub-category. The expected values must satisfy two assumptions: 1) all expected frequencies are at least one; and 2) at most 20 percent of the expected frequencies are less than five. A level of significance must be determined for the test along with the degrees of freedom for each table. The degrees of freedom are calculated by subtracting one from the number of rows and multiplying this number by the number of columns minus one. A critical value for the chi-square value is found by using a chi-square distribution table with the input values of significance level and the degrees of freedom. The chi-square value (X^2) is calculated using the formula:

$$\sum \frac{(O-E)^2}{E}$$

Where O is the observed value and E is the expected value. If the chi-square value is less than the critical value then the null hypotheses is not rejected and the two items being tested are independent of each other. If the chi-square value is more than the critical value then the null hypotheses is rejected and the two items being tested have some form of dependency on each other. [Ref. 5]

C. DATA RESULTS

1. Frequency of Class Data Results

Of the 68 different software versions reviewed in this study, 32 of them had milestone changes and 19 of them had priority one or two PTRs written on them after fleet release. This means that 47.1 percent of these software versions were delayed from their initial scheduled delivery date and 27.9 percent of them were not operationally successful. The number of software versions which were successful by both metrics (did not have a milestone change and had no priority one or two PTRs) was 31 or 45.6 percent. Further refinement of this information can be seen when it is viewed in the technical categories: software language, program size and software type.

In the area of software language, 44 (64.7 percent) of these software versions were in assembly language and 24 (35.3 percent) were in CMS-2. The number of assembly language programs which had milestone changes was 19 (43.2 percent), while six (13.6 percent) had priority one or two PTRs. A total of 22 (50.0 percent) assembly software versions passed both success metrics. Whereas, the number of CMS-2 language programs which had milestone changes was 13 (54.2 percent), while 13 (54.2 percent) also had priority one or two PTRs. A total of nine (37.5 percent) CMS-2 software versions passed both success metrics. The above data is organized in Table 1 below.

TABLE 1
SOFTWARE LANGUAGE METRIC COMPARISON

Sub-category No. / Pct	Milestone Changes No. / Pct		Priority 1 or 2 PTRs No. / Pct.		Passed Both Metrics No. / Pct.	
	Yes	No	Yes	No	Yes	No
Assembly 44 / 64.7%	19 / 43.2%	25 / 56.8%	6 / 13.6%	38 / 86.4%	22 / 50.0%	22 / 50.0%
CMS-2 24 / 35.3%	13 / 54.2%	11 / 45.8%	13 / 54.2%	11 / 45.1%	9 / 37.5%	15 / 62.5%

The category of program size had 19 (27.9 percent) software versions in the range of 0 - 90,000 bytes (small), 39 (57.4 percent) software versions in the range of 90,000 - 200,000 bytes (medium) and ten (14.7 percent) software versions in the range of 200,000 bytes and above (large). Of the 19 small-size software versions, 12 (63.2 percent) had milestone changes, eight (42.1 percent) had priority one or two PTRs and six (31.6 percent) successfully passed both metrics. Of the 39 medium-size software versions, 15 (38.5 percent) had milestone changes, seven (17.9 percent) had priority one or two PTRs written on them after fleet release and 21 (53.8 percent) successfully passed both metrics. Finally, on the ten large-size software versions, five (50.0 percent) had milestone changes, four (40.0 percent) had priority one or two PTRs written on them and four (40.0 percent) successfully passed both metrics. The above data is organized in Table 2 below.

TABLE 2
PROGRAM SIZE METRIC COMPARISON

Sub-category No. / Pct	Milestone Changes No. / Pct		Priority 1 or 2 PTRs No. / Pct.		Passed Both Metrics No. / Pct.	
	Yes	No	Yes	No	Yes	No
0 - 90,000 19 / 27.9%	12 / 63.2%	7 / 36.8%	8 / 42.1%	11 / 57.9%	6 / 31.6%	13 / 68.4%
90,000 - 200,000 39 / 57.4%	15 / 38.5%	24 / 61.5%	7 / 17.9%	32 / 82.1%	21 / 53.8%	18 / 46.2%
200,000 - ABOVE 10 / 14.7%	5 / 50.0%	5 / 50.0%	4 / 40.0%	6 / 60.0%	4 / 40.0%	6 / 60.0%

In the category of software type, seven (10.4 percent) software versions were initial, 41 (61.2 percent) software versions were upgrade, 19 (28.4 percent) software versions were maintenance and one (1.5 percent) software version was classified as other. Of the seven initial software versions, all seven (100 percent) had milestone changes and priority one or two PTRs written on them, and therefore zero (0.0 percent) passed both success metrics. Of the 41 upgrade software versions, 21 (51.2 percent) had milestone changes, five (12.2 percent) had priority one or two PTRs written on them after fleet release and 21 (51.2 percent) successfully passed both metrics. Of the 19 maintenance software versions, four (21.1 percent) had milestone changes, six (31.6 percent) had priority one or two PTRs written on them and 11 (57.9 percent) passed both metrics successfully. Finally, the one software version which was classified as other had zero (0.0 percent)

milestone changes, one (100 percent) priority one or two PTR written against it after fleet release and did not pass both success metrics. The above data is organized in Table 3 below.

TABLE 3
SOFTWARE TYPE METRIC COMPARISON

Sub-category No. / Pct	Milestone Changes No. / Pct		Priority 1 or 2 PTRs No. / Pct.		Passed Both Metrics No. / Pct.	
	Yes	No	Yes	No	Yes	No
Initial 7 / 10.4%	7 / 100%	0 / 0.0%	7 / 100%	0 / 0.0%	0 / 0.0%	7 / 100%
Upgrade 41 / 61.2%	21 / 51.2%	20 / 48.8%	5 / 12.2%	36 / 87.8%	21 / 51.2%	20 / 48.8%
Maint. 19 / 28.4%	4 / 21.1%	15 / 78.9%	6 / 31.6%	13 / 68.4%	11 / 57.9%	8 / 42.1%
Other 1 / 1.5%	0 / 0.0%	1 / 100%	1 / 100%	0 / 0.0%	0 / 0.0%	1 / 100%

Of the 32 software versions which had milestone changes, the data sources reported 68 reasons why these milestones changed. Only one of the 12 reasons listed for a milestone change in the inquiry in Appendix A was not chosen as a possible answer. The reason that did not cause a milestone change was hardware reliability. The reasons for milestone changes and their percentages of occurrence are listed in Table 4.

TABLE 4
REASONS FOR MILESTONE CHANGES

Reason	Pct	Reason	Pct
Hardware Changes	10.3%	Software Changes	17.6%
Changing User Requirements	22.0%	Software Reliability	4.4%
Budgetary Pressure	1.5%	System Integration Problems	10.3%
Inadequate Integration Time	10.3%	Political Decision	10.3%
Inadequate Design Time	1.5%	Inadequate Development time	1.5%
Other	10.3%		

2. Chi-square Data Results

The first chi-square independence test compared software language against the two metrics (milestone change and priority one or two PTRs). The null hypothesis for both tests was that software language and milestone changes (or priority one or two PTRs) are independent. The alternative hypothesis for both tests was that software language and milestone changes (or priority one or two PTRs) are dependent. The significant level used was 0.01 and the number of degrees of freedom is one which gives a corresponding critical value of 6.635. The chi-square value for milestone changes is 0.8 and for priority one or two PTRs is 12.7. Therefore, the null hypothesis for the milestone changes test is not rejected, but the null hypothesis for the priority one or two PTRs test is rejected and the alternative hypothesis is accepted. It

appears that software language is statistically independent of milestone changes but statistically dependent of priority one or two PTRs. The results of these two tests are shown below in Table 5.

TABLE 5
SOFTWARE LANGUAGE INDEPENDENCE TEST RESULTS

Sub-category	Milestone Changes Observed / Expected		Row Total	Priority 1 or 2 PTRs Observed / Expected		Row Total
	Yes	No	Yes	Yes	No	
Assembly	19 / 20.7	25 / 23.3	44	6 / 12.3	38 / 31.7	44
CMS-2	13 / 11.3	11 / 12.7	24	13 / 6.7	11 / 17.3	24
Column Total	32	36	68	19	49	68

The second chi-square independence test compared program size against the two metrics (milestone change and priority one or two PTRs). The null hypothesis for both tests was that program size and milestone changes (or priority one or two PTRs) are independent. The alternative hypothesis for both tests was that program size and milestone changes (or priority one or two PTRs) are dependent. The significant level used was 0.01 and the number of degrees of freedom is two which gives a corresponding critical value of 9.21 for each test. The chi-square value for milestone changes is 3.2 and for priority one or two PTRs is 4.6. Therefore, the null hypothesis for both tests is not rejected, and it appears that program size is statistically independent of milestone changes

and priority one or two PTRs. The results of these two tests are shown below in Table 6.

TABLE 6
PROGRAM SIZE INDEPENDENCE TEST RESULTS

Sub-category	Milestone Changes Observed / Expected		Row Total	Priority 1 or 2 PTRs Observed / Expected		Row Total
	Yes	No	Yes	Yes	No	
Small	12 / 8.9	7 / 10.1	19	8 / 5.3	11 / 13.7	19
Medium	15 / 18.4	24 / 20.6	39	7 / 10.9	32 / 28.1	39
Large	5 / 4.7	5 / 5.3	10	4 / 2.8	6 / 7.2	10
Column Total	32	36	68	19	49	68

The third chi-square independence test compared software type against the two metrics (milestone change and priority one or two PTRs). Since there was only one software version of software type other, this single value was deleted from this test. Therefore, these two tests will only have 67 values instead of 68 as the previous four test have had. The null hypothesis for both tests was that software type and milestone changes (or priority one or two PTRs) are independent. The alternative hypothesis for both tests was that software type and milestone changes (or priority one or two PTRs) are dependent. The significant level used was 0.01 and the number of degrees of freedom is two which gives a corresponding critical value of 9.21 for each test. The chi-

square value for milestone changes is 13.3 and for priority one or two PTRs is 23.8. Therefore, the null hypothesis for both tests is rejected, and the alternative hypothesis is accepted. It appears that software type is statistically dependent of milestone changes and priority one or two PTRs. The results of these two tests are shown below in Table 7. It should be noted that the results for the chi-square independence test for software type and milestone changes may not be totally valid, since one of the chi-square test assumptions as stated in chapter IV Section B subsection 2 above has been violated.

TABLE 7
SOFTWARE TYPE INDEPENDENCE TEST RESULTS

Sub-category	Milestone Changes Observed / Expected		Row Total	Priority 1 or 2 PTRs Observed / Expected		Row Total
	Yes	No	Yes	Yes	No	
Initial	7 / 3.3	0 / 3.7	7	7 / 1.9	0 / 5.1	7
Upgrade	21 / 19.6	20 / 21.4	41	5 / 11.0	36 / 30.0	41
Maintenance	4 / 9.1	15 / 9.9	19	6 / 5.1	13 / 13.9	19
Column Total	32	35	67	18	49	67

The fourth chi-square independence test compared the two metrics (milestone change and priority one or two PTRs) against each other. The null hypothesis was that milestone changes and priority one or two PTRs are independent. The

alternative hypothesis was that milestone changes and priority one or two PTRs are dependent. The significant level used was 0.01 and the number of degrees of freedom is one which gives a corresponding critical value of 6.635 the test. The chi-square value for milestone changes is 7.5. Therefore, the null hypotheses for the test is rejected, and the alternative hypothesis is accepted. It appears that milestone changes are statistically dependent of priority one or two PTRs. The results of this test are shown below in Table 8.

TABLE 8
METRICS INDEPENDENCE TEST RESULTS

Observed / Expected		Priority 1 or 2 PTRs		Row Total
		Yes	No	
Milestone Changes	Yes	14 / 8.9	18 / 23.1	32
	No	5 / 10.1	31 / 25.9	36
Column Total		19	49	68

V. CONCLUSION

This research has produced many interesting results. From the raw collected data to conversations with software developers, these results express the facts and opinions of Naval aviation software developers with respect to their specific aircraft platform. This study has compiled these results to present an overall view of Naval aviation software development.

A. EXPLANATION OF RESULTS

1. Explanation of Software Language Results

As shown in Table 1, the language (assembly or CMS-2) that a software version is written in only has a significant difference in the priority one or two PTRs metric. CMS-2 software versions being released to the fleet have a four times greater percentage in having priority one or two PTRs as assembly language versions.

The chi-square independence test results of Table 5 have shown that software language and milestone changes are statistically independent, while software language and priority one or two PTRs are statistically dependent.

2. Explanation of Program Size Results

The results in Table 2 also show that the size of the software program had no significant affect on the success of

a software version against the metrics. However, medium-size (90,000 - 200,000) software versions on average do slightly better against the metrics than large-size (200,000 and above) software version which do slightly better than small-size (0-90,000) software versions.

The results from the chi-square independence test of Table 6 verifies that program size is statistically independent of the occurrence milestone changes and priority one or two PTRs.

3. **Explanation of Software Type Results**

The most significant result of this study is in the category of software type. As shown in Table 3, ALL initial software versions had to change their original milestone schedule, and when finally released to the fleet, they had major problems that had to be reworked.

Table 3 also shows that upgrade software versions are two times more likely to have a milestone change as maintenance versions, but less than half as likely to cause priority one or two PTRs to be generated after the software is released to the fleet. However, both maintenance and upgrade software versions are approximately equal in passing both metrics (no milestone changes and no priority one or two PTRs).

The results of the chi-square independence tests of Table 7 show that type of software version affects the

occurrence of whether a software version is delayed or will have problems after fleet release.

4. **Explanation of Reasons for Milestone Change Results**

The reasons for milestone changes, as summarized in Table 4, show that software reliability, budgetary pressure, inadequate design time and inadequate development time are rarely a reason for changing a milestone since all four reasons together account for only 9.9 percent of the problems. The most prominent reason for milestones to change is because of changing user requirements, which accounted for 22 percent of the changes. The second most prominent reason for milestones to change is because of software changes, which accounted for 17.6 percent of the changes. Hardware changes, system integration problems, internal/external political decisions and miscellaneous other causes (usually dealing with documentation) were each the reason for milestone changes 10.3 percent of the time.

5. **Explanation of Software Metrics Results**

The results of the chi-square independence tests of Table 8 have shown that whether a software version is delayed or not does not affect the probability that the software version will have major reported problems after it is released to the fleet.

B. SUMMARIZING THE RESULTS

The most important success factor in defining a successful software version is the category of software type. This factor was further confirmed with the results of the chi-square independence test, since software type was the one category which had a statistical dependence of milestone changes and priority one or two PTRs.

"Maintenance" types of software versions are the most successful (57.9 percent) in staying on original schedule and being trouble-free after release to the fleet. However, medium-size software versions, "upgrade" types of software versions and assembly language software versions are 50 percent or better at passing both metrics.

In contrast, "Initial" types of software versions were never able to maintain original schedule or be released to the fleet without major problems being discovered afterwards. All other subcategories (excluding the software type subcategory of other) are nearly equal in their percentage (a narrow range between 30 - 40 percent) of successfully passing both metrics. The software subcategories are ranked by their success at passing both metrics in Table 9 below.

Further analysis of the results has shown that of the 15 software versions which had milestone changes due to "changing user requirements" (the reason cited most often for a milestone change), 12 of these changes occurred in upgrade type software versions. This result is not totally unexpected

since upgrade software versions, in an effort to enhance fleet user capabilities as much as possible, try until the very last minute to add the latest requests for new system features. Maintenance versions on the other hand are usually more stable since they are trying to correct problems of a current software version, and few new functions are normally added. [Ref. 6]

TABLE 9
RANKING OF SOFTWARE SUBCATEGORIES BY PERCENTAGE

Subcategory	Percentage that Passed Both Metrics
Maintenance Type Versions	50.0%
Medium Size Programs	50.0%
Upgrade Type Versions	50.0%
Assembly Versions	50.0%
Large Size Programs	40.0%
CMS-2 Versions	40.0%
Small Size Programs	40.0%
Initial Type Versions	0.0%
Other Type Versions	0.0%

C. RECOMMENDATIONS

As previously discussed, the category of software type produced some very noteworthy results and showed possible areas where improvements could be made. This section notes some of the shortfalls discovered in this study and recommends some solutions.

Situation ONE

The inability of the initial software versions to be produced without having to change their original milestone schedules.

Recommendation - For initial versions where software is being developed for the first time for a new aircraft configuration or where computer hardware is being added and/or changed for an existing aircraft system, more time is needed in the development process. Changes could possibly be made in the method used to calculate software development time schedules and allow for more development time for original versions of a software program. To some extent this extra time could be used to better define the system specifications or ensure integration problems are more thoroughly worked out. It would provide a more accurate implementation schedule. Further research to determine a more exact method for calculating software development time could more fully define a list of factors which cause schedule delays.

Situation TWO

The most significant result was that even after changing their schedules these initial software versions had serious software problems which were not discovered until the software was in the fleet. For instance, a software version was released to the fleet after successfully passing testing, and under normal flight situations the computer would stop working.

Recommendation

The best solution to this problem is to ensure specifications are thoroughly defined and that all areas of testing are well specified and properly accomplished.

Situation THREE

The low success rate of upgrade software versions (48.8 percent) compared to maintenance software versions (78.9 percent) in being able to maintain the original development schedule needs to be increased.

Recommendation

The suggested solution to revamp this problem is similar to situation one above. More time is needed in the development schedule. This in turn requires a more accurate method for estimating the upgrade schedule. Consideration should be given to solidifying software specifications when originally planned and not allowing any new changes or enhancements to be added to this baseline. New changes or enhancements would be handled in future updates.

If an urgent change or enhancement is needed "NOW", this change should be made to the current working software version. If determined to be of a less urgent status, then add it to the follow-on version to the current software under development. Adding a late change or enhancement to an already baselined software version only causes problems in the

development schedule. The later such a change is made the more costly and time consuming the problem becomes. [Ref. 7]

Situation FOUR

As noted in Chapter IV Section C, on the average 45.6 percent of the software versions passed both metrics (no milestone changes and no priority one or two PTRs). This average needs to be raised. An initial goal of at least 50 percent should be made. In keeping with the DOD implementation of Total Quality Leadership (or continuous process improvement) efforts should continue to improve in this area.

Recommendation

Applying the recommendations of situations 1, 2 and 3 can help improve this percentage. Also, a more thorough study of this specific problem could generate procedures which would improve the entire software development process for DOD and save the government time and money.

D. OVERVIEW OF THE DOD SOFTWARE DEVELOPMENT PROCESS

The DOD standard for software development, DOD-STD-2167A, and the entire software development process are well established, especially considering today's knowledge of this process. However, software development is neither a science nor a strict engineering discipline. It is more like an art, and is difficult to manage. [Ref. 8]

The DOD situation is further compounded because it endeavors to develop software for computers whose use is being continuously updated or completely changed. When the DOD is developing software for an embedded aircraft computer system, it is not like developing software for a personnel computer (PC). Most aircraft computers are real-time or near real-time systems. The software in these aircraft must respond to numerous inputs and produce several outputs instantaneously without failure. If the software in a PC fails, the user can restart the computer and try the problem again. If an aircraft is in a combat situation or needs the software for aircraft control, the crew may not have time to restart the computer and this may cost the government the lose of an aircraft and a crew. As a result, military software has an important requirement for minimal or zero software faults.

For its part, the DON as a whole does a remarkable job of managing and producing software for embedded aircraft computer systems. In developing aircraft software, the DON must take into account a continuously changing world situation and the bureaucracy of the appropriation process for receiving funding. Additionally, all aircraft missions and capabilities are different, and each aircraft type must have software developed for it that will integrate correctly with unique hardware and avionic systems. However, in the process of collecting data for this study the following points were noted:

1. Relationship between software developer and technical and operational testers.

- The software developers work hard to give the fleet user the best possible product with the newest technology and features as quickly as possible.

- The testing agencies want to give the fleet a high quality product. They strive for zero defects in the software and work to ensure that the product is capable of doing the mission it was designed for.

It would seem that both organizations, the software developers and the testing community, are working for the same thing, a successful product for the fleet. However at times, developers believe that not all technical and operational testing is required for every version of software; testers believe that any change to a software version should go through some if not all forms of testing. The developer's opinion is that testing will delay a good software product from being delivered quickly to the fleet. The tester's opinion is that if a bad software product is delivered to the fleet then the fleet is going to be less capable than before. A defective new software version is even further delayed.

The data collected for this study show that there have been software products which have successfully passed the testing process without problems, ones in which problems were found and returned for corrections, and those that were sent out to the fleet with problems that were later discovered.

This area of "when" or "if" a software version should be tested is important enough to warrant a study of the situation to see if the current process should be revamped. However, the bottom line is that fleet user will be the one who decides if the product can be used effectively to get the mission accomplished because product environments change and what was useful yesterday may not be correct for the situation today.

2. Incorporation of ADA.

ADA is the High Order Language (HOL) used for computer programming that DOD had developed in an effort to standardize computer programming, logistics and support from several languages to one software language. For the Navy, OPNAVINST 5200.28 has mandated that "Ada is required for new developments and shall be phased into use for existing systems at the next major upgrade." [Ref. 9] Congress, in the fiscal year 1991 budget, mandated ADA to be used [Ref. 10].

There are two potential problems the Navy has in incorporating ADA into existing systems. First, ADA is a relatively new software language and as such the programming experience level of ADA programmers is small. Second and perhaps most important, all the current software engineers that work to develop software for Navy aircraft and have many years of experience in developing aviation software, have little or no experience with ADA. This study has shown all aircraft computer programming for the programs reviewed is in

either assembler or CMS-2. The Navy needs to establish a plan which considers any or a combination of the following points:

1) provide the experienced software engineers training in the ADA language. They are valuable resources having developed their aircraft software for many years.

2) all newly hired programmers should be trained in ADA.

3) be prepared to incur the additional learning curve transition to ADA which must be figured into the development schedule.

A thorough study and analysis of this situation will provide valuable information and alternatives to make the optimum decision.

3. PTR reporting.

The use of program trouble reports for reporting software or system problems by the fleet user may be lacking for the following reasons:

a. the aircraft crews may find a way to work-around a problem or discrepancy, but in so doing are adapting themselves to the discrepancy situation rather than making sure the software or system is performing as specified. Because the aircraft crew has discovered a suitable work-around for the problem, a PTR may not be written, and the discrepancy is not reported.

b. all aircraft crews may not know how to report software or system problems, or do not believe it is important

to take the time to write down a discrepancy if a suitable work-around can be used instead. This situation of aircraft crews being uninformed about the importance of PTRs can lead to numerous software problems that are not reported and later may cause more serious problems.

4. Software testing.

In the collection of data for this research, software developers have expressed the importance of thorough validation and testing. The use of independent validation and verification (IV&V) and stress testing to discover major software problems is essential. Lack or only partial completion of these two forms of testing have been a major factor in software being released for technical and operational testing with priority one and two deficiencies.

5. Software Integration.

The integration of computer hardware with the software that will be operating on it is always a factor software developers take into consideration. However, some of the reasons for software delay and priority one or two PTRs are from integration problems with other aircraft systems. The source of these integration problems come from new or updated weapon systems (including weapon ballistics), auxiliary computers, or other avionics systems. The cause of these integration problems is normally that the change to the other aircraft system is considered by its developer to be so minute that this change should not affect any other system. This

unfortunately is not always the case. The important key is communication between developers is needed when a change is made to a system that integrates with a computer.

E. POSSIBLE FOLLOW-ON TOPICS

The area of mission critical computer software for Naval aviation is critical for the future. Further in-depth research beyond this study can greatly assist with future decisions on Naval aviation software development. The following suggested research topics can provide valuable information for making these decisions.

1. A thorough study of individual aircraft platform's software. From when the initial software requirements were made with the software developer for the original aircraft through the software life cycle to the current version.

2. A detailed study of the entire software maintenance process and documentation of what NAVAIR, the Software Support Activities (SSAs) and the testing agencies do to make changes in weapon system software.

3. A more in-depth study of why initial software versions have problems maintaining original software development schedule and even when their schedules are updated major software problems still occur (i.e., priority one or two PTRs).

4. Research to determine why software version schedules fail to be met in excess of 50 percent. Also why these

programs are delivered with an inordinately high rate of major errors discovered after fleet release.

5. An examination on the amount of testing that is statistically required for a software version after it has left the SSA (i.e. does it need TechEval and/or OT&E).

6. Analysis to determine what affect ANSI/MIL-STD-1815A (Reference Manual for the ADA Programming Language) and its implementation directives will have on the software development process and the software developers (contractors and the SSAs) since most of the programmers know either assembler or CMS-2.

7. Thorough study of the ability of the software developers to deliver software versions within budget. This study could be combined with the data from this thesis which would produce the more classic software study of the software developers ability to meet cost and schedule requirements and the difficulties in meeting these two criteria.

8. Incorporation of the results of any of these research topics into a decision support system (DSS). A DSS could assist program managers or software developers in making decisions in many areas of the software development process. These systems would improve the entire Naval aviation software development process.

F. LESSONS LEARNED

The following lessons learned were noted during the entire life cycle of this research:

1. To gain an understanding and appreciation of the area of research, the researcher must be immersed into the research environment. This may entail asking the wrong or foolish questions, but later on this will enable the researcher to ask the right questions and collect the correct data.

2. Questionnaires are adequate for data collection but face-to-face interviews are a faster way to collect data. Additionally, in-person interviews have the added advantage of allowing the researcher to get a better understanding of the data environment.

3. For all forms of data collection, allow ample time for personnel to respond to research questions, but set a FIRM last day for acceptance of data collection and stick to it. Direct follow-up will ensure a higher response rate.

4. The researcher should make the original research objective realistic yet flexible. This allows the researcher to modify the objective for contingencies such as needed data is not easily accessible in a timely fashion or not available at all.

G. FINAL THOUGHTS

This study has only scratched the surface for evaluating the software development process of different types of

aircraft. To gain a more thorough understanding of the successes and failures of the Naval aviation software development process, further research that concentrates specifically on each aircraft type is needed. This in-depth research will help to correct problems each development process has, while allowing all other aircraft types to benefit from their successes.

DOD-STD-2167A was established with the intention of allowing each aircraft program enough flexibility to develop mission critical software under any feasible development method, while still furnishing an architecture to work with. However, all phases of the software development method that is selected must be thoroughly implemented otherwise milestone delays and major deficiencies in fleet released software occur. Most problems in Naval aviation software development seem to occur when user requirements change after software development has commenced or when a development phase is not completely performed (e.g. incomplete stress testing).

With decreasing budgets, Naval aviation software development must become as efficient as possible. This will require improvements in all areas of software development and the overall commitment of everyone involved in this process to a total and integrated team or mission concept. This task will be difficult, but with the implementation of Total Quality Leadership, it will not be impossible.

APPENDIX A

INQUIRY FOR DATA COLLECTION

1. What type of aircraft is the computer software used for? (circle all applicable platforms)

A. A-6 B. AV-8

C. EA-6 D. E-2

E. F-14 F. F-18

G. P-3 H. S-3

I. SH-2/3 J. SH-60B

K. SH-60F

L. Others (please specify)_____

2. What type of computer system is the software to be used for? (circle all applicable computer systems)

A. AN/AYK-10 B. AN/AYK-14

C. ASN-123 D. ASN-150

E. CP-3B F. CP-901

G. TDY-43

H. Others (please specify)_____

3. What is the total number of lines of code in the software program today and what software languages is it written in?

4. Please provide a copy of the Software Life Cycle Schedule (Milestone charts) for as much of the software program history as is available (i.e. from the initial software program to the current fleet release). If the above is not possible, please annotate when and what Milestones were revised for each version of software.)

5. Using the Milestones in number 4 above, please list what Milestones were changed and why they needed to be revised from the previous estimate. (Note: a Version number or software baseline designator are considered the same.)

Possible Answers May Be:

A. Hardware changes	B. Software changes
C. Changing user requirements	D. Hardware reliability
E. Software reliability	F. Budgetary pressures
G. System integration problems	H. Inadequate integration time
I. Political decision	J. Inadequate design time
K. Inadequate development time	L. Others (specify below)

A. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

B. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

C. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

D. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

E. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

F. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

G. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

H. Version number _____ Milestone _____

Reason(s) for change (circle all appropriate answer(s)):

A B C D E F G H I J K L

Other(s) (please specify) _____

_____

6. For each version of software in number 4 above, what is the number of lines of newly written or changed source code for each version of software from previous baseline?

A. Version number _____ Number of lines _____

B. Version number _____ Number of lines _____

C. Version number _____ Number of lines _____

D. Version number _____ Number of lines _____

E. Version number _____ Number of lines _____

F. Version number _____ Number of lines _____

G. Version number _____ Number of lines _____

H. Version number _____ Number of lines _____

I. Version number _____ Number of lines _____

J. Version number _____ Number of lines _____

7. For each version of software in number 4 above, what Type would you classify it as?

Possible Answers are:

A. Initial release

B. Major upgrade release

C. Maintenance release to fix priority 1 or 2 PTRs.

D. Other (please specify)

A. Version number _____ Type _____

B. Version number _____ Type _____

C. Version number _____ Type _____

D. Version number _____ Type _____

E. Version number _____ Type _____

F. Version number _____ Type _____

G. Version number _____ Type _____

H. Version number _____ Type _____

I. Version number _____ Type _____

J. Version number _____ Type _____

8. Were Priority 1/Priority 2 Problem Reports or Emergency PTRs written against any of the software versions within 3 years of Fleet Issue? NO / YES (circle one)

If YES, please elaborate on what problem the software program had and how the problem was fixed.

A. Software version number _____

Problem _____

_____

_____

Solution _____

_____

_____

B. Software version number _____

Problem _____

_____

_____

Solution _____

_____

_____

C. Software version number _____

Problem _____

_____

_____

Solution _____

_____

_____

D. Software version number _____

Problem _____

_____

_____

Solution _____

_____

_____

E. Software version number _____

Problem _____

_____

_____

Solution _____

_____

_____

F. Software version number _____

Problem _____

_____

_____

Solution _____

_____

_____

9. Please include any other comments which may be of use for this software analysis.

APPENDIX B

LIST OF ACRONYMS

Chief of Naval Operations Instruction	OPNAVINST
Commander, Space and Naval Warfare Systems Command	COMSPAWARSYSCOM
Computer Software Component	CSC
Computer Software Configuration Items	CSCI
Computer Software Unit	CSU
Configuration Item	CI
Critical Design Review	CDR
Department of Defense	DOD
Department of the Navy	DON
Embedded Computer	EC
Embedded Computer Resources	ECR
Engineering Change Proposals	ECPs
Formal Qualification Testing	FQT
Full Scale Engineering Development	FSED
Functional Configuration Audit	FCA
High Order Language	HOL
Independent Validation and Verification	IV&V
Interface Design Document	IDD
Lines of Code	LOC
Management Steering Committee for Embedded Computer Resources	MSC-ECR
Mission-Critical Computer Resources	MCCR
Naval Aviation Command	NAVAIR
Naval Electronic Systems Command	NAVELEX

Operational Flight Program	OFP
Personnel Computer	PC
Physical Configuration Audit	PCA
Post-Deployment Software Support	PDSS
Preliminary Design Review	PDR
Program Managers	PMs
Program Trouble Reports	PTRs
Programmable Calculators	PROCALS
Research, Development and Acquisition	RDA
Secretary of the Navy Instruction	SECNAVINST
Software Design Document	SDD
Software Support Activities	SSAs
Software Test Description	STD
Software Test Plan	STP
Software Test Report	STR
Software Trouble Reports	STRs
Tactical Digital Standards	TADSTANDS
Test Readiness Review	TRR

LIST OF REFERENCES

1. Glaseman, S., *Comparative Studies in Software Acquisition*, pp. 4-8, Lexington Books, 1982.

2. Kitfield, J., "Is Software DOD's Achilles' Heel?," *Military Forum*, pg. 29, July 1989.

3. Department of Defense Military Standard DOD-STD-2167A, *Defense System Software Development*, 29 February 1988; Department of Defense Military Standard MIL-STD-1521B (USAF), *Technical Reviews and Audits for Systems, Equipments, and Computer Software*, 4 June 1985; Department of Defense Military Standard MIL-STD-480B, *Configuration Control - Engineering Changes, Deviations and Waivers*, 15 July 1988; Department of the Navy Tactical Digital Standard A, *Standard Definitions for Embedded Computer Resources in Tactical Digital Systems*, 2 July 1980; Department of the Navy Tactical Digital Standard D Revision 1, *Reserve Capacity Requirements for Mission-Critical Systems*, 27 October 1989.

4. Hassett, M. J., and Weiss, N. A., *Introductory Statistics*, 2d ed., pg. 160, Addison-Wesley Publishing Company, 1989.

5. Hassett, M. J., and Weiss, N. A., *Introductory Statistics*, 2d ed., pp. 473-481, Addison-Wesley Publishing Company, 1989.

6. Jones, C., *Programming Productivity*, pp. 141-142, McGraw-Hill, Inc., 1986.

7. *Mission Critical Computer Resources Management Guide*, pg. 7-7, Defense Systems Management College, Fort Belvior, VA., 1990.

8. Boehm, B. W., and Ross R., "Theory-W Software Project Management: Principles and Examples," *IEEE Transactions on Software Engineering*, v.15, no.7, pg. 902, 7 July 1989.

9. Chief of Naval Operations Instruction 5200.28, *Life Cycle Management of Mission-Critical Computer Resources (MCCR) for Navy Systems Managed Under the Research, Development, and Acquisition (RDA) Process*, 25 September 1986.

10. Schwartz, K. D., "Ada Use Is Mandatory As of June," *Government Computer News*, pg 1, 10 December 1990.

BIBLIOGRAPHY

Attanasio, H., *Contracting for Embedded Computer Software within the Department of the Navy*, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1990.

Boehm, B. W., "Improving Software Productivity," *COMPUTER*, September 1987.

Boehm, B. W., "Software Engineering Economics," *IEEE Transactions on Software Engineering*, v. SE10, no. 1, 1 January 1984.

Brooks, F. P., Jr., "No Silver Bullet - Essence and Accidents of Software Engineering," *Information Processing 86*, 1986.

Brooks, F. P., Jr., *The Mythical Man-Month*, Addison-Wesley, Publishing Company, 1975.

Burke, J. D., Capt, USA, "Software Testing Management," *Program Manager*, May-June 1988.

Department of Defense Directive 5000.29, *Management of Computer Resources in Major Defense Systems*, 26 April 1976.

Department of Defense Instruction 5000.31, *Interim List of DoD Approved High Order Programming Languages (HOL)*, 24 November 1976.

Department of Defense Military Handbook MIL-HDBK-287, *A Tailoring Guide for DOD-STD-2167A, Defense System Software Development*, 11 August 1989.

Department of Defense Military Standard DOD-STD-1679A (NAVY), *Software Development*, 22 October 1983.

Department of Defense Military Standard ANSI/MIL-STD-1815A-1983, *Reference Manual for the Ada Programming Language*, 17 February 1983.

Department of Defense Military Standard DOD-STD-2168, *Defense System Software Quality Program*, 1 April 1985.

Department of the Navy, Naval Electronics Systems Command Instruction (NAVELEX INST) 5200.23, *NAVELEX Computer Software Life-Cycle Management Guide*, 17 December 1979.

Department of the Navy, Secretary of the Navy Instruction (SECNAVINST) 5200.32, *Management of Embedded Computer Resources in Department of the Navy Systems*, 11 June 1979.

Department of the Navy, Tactical Digital Standard (TADSTANDS) B, *Standard Embedded Computers, Peripherals, and Input/Output Interfaces*, 5 March 1990.

Department of the Navy, Tactical Digital Standard (TADSTANDS) C, *Computer Programming Language Standardization Policy for Mission-Critical Computer Resources*, 15 August 1990.

Department of the Navy, Tactical Digital Standard (TADSTANDS) E, *Software Development, Documentation, and Testing Policy for Navy Mission Critical Systems*, 24 January 1989.

Deutsch, M. S., "An Exploratory Analysis Relating the Software Project Management Process to Project Success." Working Paper for Hughes Aircraft Company, 1990.

Fritz, R. L. and Shocket, F., "LAMPS: Demonstrated Maintainability through Application of MIL-SPEC Software Development Techniques," paper presented at the Conference on Software Maintenance-1988, Phoenix, Arizona, October 24-27, 1988.

Lientz, B. P., and Swanson, E. B., "Problems in Application Software Maintenance," *Communications of the ACM*, November 1981.

Scacchi, W., "Understanding Software Productivity: A Comparative Empirical Review," *IEEE*, 1989.

Subcommittee on Investigations and Oversight, U.S. House of Representatives, *Bugs In the Program", Problems In Federal Government Computer Software Development and Regulation*, Washington, D.C., September 1989.

Texas Instruments, Inc., Technical Report RCI-TR-012, *A Poor Man's Guide To Estimating Software Costs*, by D. J. Reifer, 1 November 1985.

United States General Accounting Office, Information Management and Technology Division, *EMBEDDED COMPUTERS: Navy's Approach to Developing Patrol Aircraft Avionics System Too Risky*, September 1990.

INITIAL DISTRIBUTION LIST

1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Martin J. McCaffrey AS/Mf Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
4.	Dr. Tarek Abdel-Hamid, AS/Ah Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
5.	Program Executive Officer for Space, Communications and Sensors PMW-148 2511 Jefferson Davis Hwy Washington, D.C. 20363-5100 Attn: LCDR Robert L. Buckley	1
6.	Commander Naval Weapons Center Code 3104 China Lake, California 93555	1
7.	Commander Naval Weapons Center Code 3103A China Lake, California 93555	1
8.	Commander Naval Weapons Center Code 3193 China Lake, California 93555	1
9.	Commander Naval Weapons Center Code 3103B China Lake, California 93555-6001	1
10.	Commander Naval Weapons Center Code 3107 China Lake, California 93555-5000	1
11.	Commander Pacific Missile Test Center Pt. Mugu, California 93042-5000 Attn: Code 4060A	1

12. Commander Pacific Missile Test Center 1
Code 9040.1
Pt. Mugu, California 93042-5000

13. Fleet Combat Direction Systems Support Activity 1
200 Catalina Blvd
San Diego, California 92147-5081
Attn: Code 3

14. Commander Naval Air Development Center 1
Street Road
Warminster, Pennsylvania 18974-5000
Attn: Code 103J1

15. Commander Naval Air Development Center 1
Street and Jacksonville Road
Warminster, Pennsylvania 18974
Attn: Code 101C

16. Commanding Officer 1
Naval Aviation Depot
Code 331
NAS North Island
San Diego, California 92135-5112
Attn: Ken Pecus

17. Commanding Officer 1
Naval Air Development Center
Warminster, Pennsylvania 18974
Attn: Code 1022

18. Department of the Navy 1
Naval Air System Command
Washington, D.C. 20361-5460
Attn: AIR-54661

19. Department of Defense 1
Defense Systems Management College
Fort Belvior, Virginia 22060-5426
Attn: Mr. Alan Roberts

20. Michael S. Deutsch 1
Hughes Aircraft Company
Ground Systems Group
P. O. Box 3310
Fullerton, California 92634

Thesis
B83185 Buckley
c.1 An analysis of mission
critical computer soft-
ware in Naval aviation.